créer un compte Connexion

page discussion modifier historique

navigation
- Accueil
- Communauté
- Actualités
- Modifications récentes
- Page au hasard
- Aide

rechercher
Lire Rechercher

boîte à outils
- Pages liées
- Suivi des pages liées
- Pages spéciales
- Version imprimable
- Adresse de cette version

google adsense

partager

Les commandes fondamentales de Linux

Sommaire

1 Introduction
1.1 Contributions et remerciements
1.2 Objectifs
1.3 Manœuvres préliminaires
1.3.1 Émulateur
1.3.2 Live
1.3.3 Installation
2 Composition du système
3 Commandes fondamentales
3.1 Mise en place
3.1.1 Connexion
3.2 Arrêter le système (**shutdown**)
3.3 Variables d'environnement (echo)
3.4 Se déplacer dans l'arborescence de répertoires (**cd**)
3.5 À quel programme correspond une commande ? (type)
3.6 Dans quel répertoire suis-je actuellement ? (**pwd**)
3.7 Lister le contenu d'un répertoire (**ls**)
3.7.1 Motifs (glob)
3.7.2 Options
3.8 Voir le contenu d'un fichier (**less**)
3.9 Modifier le contenu d'un fichier (l'éditer) (**emacs**, **joe**)
3.9.1 joe
3.9.1.1 Configuration
3.9.2 Emacs
3.9.3 joe *est* Emacs
3.10 Créer un compte utilisateur
3.10.1 Préambule
3.10.1.1 Invite (prompt) de root ou d'utilisateur
3.11 Déclarer qui est propriétaire d'un fichier (**chown**)
3.12 Copier un fichier ou un répertoire (**cp**)
3.12.1 Options
3.12.1.1 **-r**
3.13 Déclarer les droits d'accès (permissions) accordés (**chmod**)
3.13.1 Expression numérique des permissions (système octal)
3.14 Supprimer un fichier (*rm*)
3.14.1 Options
3.15 Créer un répertoire (**mkdir**)
3.16 Déplacer ou renommer un fichier (**mv**)
3.16.1 Options
3.17 Chercher un fichier ou répertoire (**find**, **locate**, **which**)
3.17.1 **find**
3.17.1.1 -type
3.17.1.2 -exec
3.17.2 **locate**
3.18 Trouver un texte (**grep**)
3.19 Compactage et décompactage (**compress**, **gzip**, **bzip2**, **7zip**)
3.19.1 **compress**
3.19.2 **gzip**
3.19.3 **bzip2**
3.19.4 **7zip**
3.20 Archivage de données (**tar**)
3.21 Connaître l'espace disque utilisé (**df** et **du**)
3.21.1 Options
3.22 Contrôler les ressources utilisées par les processus (**top**, **ps**, **kill**)
3.22.1 **top**
3.22.2 **ps**
3.22.2.1 **pstree**
3.22.3 **kill**
3.23 Redirection (des entrées/sorties d'exécutables)
3.24 Interconnexion de commandes: les pipes
3.25 Lien (**ln**)
3.26 Alias (**alias**)
4 Pour aller plus loin

Introduction [modifier]

- Premier auteur (jusqu'à 1998): Armand Delcros
- Wikisé par (depuis 2009): Nat Makarevitch

Contributions et remerciements [modifier]

- Nat Makarevitch pour ses nombreuses critiques et corrections,
- Thomas Ortega pour ses bienveillantes remarques et la remise en forme HTML du document,
- Kim Ashue pour sa relecture,
- Philibert de Mercey pour son aide à la mise en page.

L'objectif de ce document est d'enseigner les commandes fondamentales de Linux (et donc d'Unix).

Objectifs [modifier]

Nous explorerons ce qu'offre le shell le plus utilisé sous Linux, donc une partie de ce que peut faire un utilisateur capable de diriger la machine en créant des lignes de commandes plutôt qu'en se contentant d'un cliquodrôme.

Manœuvres préliminaires [modifier]

Pour apprendre il faut pratiquer, donc disposer d'une machine fonctionnant sous Linux qui n'est pas sérieusement exploitée. Son arrêt ou sa modification ne doit rien menacer.

Si vous en avez déjà une négligez la présente section (passez à la suivante).

Émulateur [modifier]

Un émulateur animant Linux sous votre navigateur web est le moyen le plus simple de commencer. Votre navigateur simulera une machine qui exécutera Linux. Pour cela cliquer ici , ce qui doit ouvrir un nouvel onglet où vous trouverez une machine virtuelle fonctionnant sous Linux, immédiatement utilisable. Elle fonctionne dans votre navigateur et ne peut rien casser. N'hésitez pas à explorer et à expérimenter car il n'y a aucun risque d'endommager quoi que ce soit.

Certaines commandes n'y sont pas installées mais c'est le moyen le plus facile, immédiat et sans danger d'aborder le sujet.

Il faut toutefois employer un navigateur récent et un ordinateur au processeur puissant.

Live [modifier]

Si vous disposez d'une machine il n'est pas nécessaire d'y installer d'emblée Linux, il suffit dans un premier temps de l'amorcer de façon « live ».

Vous devrez graver un CD ou un DVD.

Nous recommandons une version de Linux (« distribution ») appelée Knoppix .

Télécharger tout d'abord l'image de CD ou de DVD, par exemple (en France Métropolitaine) de Free.fr ou (en anglais) via le document officiel . Il est recommandé de vérifier à tout le moins son empreinte SHA et, au mieux, GPG, toutefois cela dépasse le cadre du présent document.

Graver le disque correspondant.

Amorcer la machine grâce à ce disque. Une invite libellée « boot: » apparaîtra (en bas). Saisir « knoppix lang=fr ». Si vous n'y parvenez pas attendre démarrage complet puis, ouvrir un « terminal » et saisir « setxkbmap 'fr(multi)' » (la documentation officielle n'existe qu'en anglais).

Linux fonctionne alors sans être installé, toutefois il est possible de modifier le contenu des disques de la machine donc ne pas faire n'importe quoi.

Installation [modifier]

Si vous le pouvez installez Linux sur une machine, de préférence virtuelle (VMware, Microsoft Virtual PC...).

Nous recommandons la distribution utilisée par ceux qui vous aideront, ou à défaut appelée Ubuntu ou Debian .

Composition du système [modifier]

« Linux » est le nom du seul noyau, le cœur du système d'exploitation qui est lui-même un ensemble de logiciels grâce auxquels la machine est rendue plus facilement utilisable par l'humain comme par une application.

Une famille de logiciels appelés shells fournit un moyen de diriger ce dernier en lui communiquant des commandes. Nous exposerons ici la façon d'employer le shell le plus répandu sous Linux, nommé « Bash », ainsi que certains utilitaires.

Commandes fondamentales [modifier]

ATTENTION : SOUS Linux, comme sous tout système UNIX, minuscules ET MAJUSCULES NE SONT PAS ÉQUIVALENTES.

Mise en place [modifier]

La plupart des distributions Linux démarrent d'emblée, sitôt le système amorcé, une interface graphique fournie par un ensemble logiciel appelé X Window et donnant à l'écran d'accueil une apparence agréable dont voici un exemple (cliquer sur l'image pour zoomer):

Si X Window n'est pas démarré vous ne verrez aucune fenêtre et surtout des caractères, quasi pas de graphisme sauf peut-être un Tux en haut à gauche, du charabia terminé par la mention « login: » suivie d'un curseur clignotant:

Connexion [modifier]

(Cette étape n'est pas nécessaire si vous employez l'émulateur préconisé)

Connectez-vous au système en tant qu'utilisateur root, donc saisir *root* en guise de «login» (et soumettre grâce à la touche « Entrée », évidemment!), puis le mot de passe («password») adéquat (qui a été défini lors de l'installation ou est fourni par la documentation).

Si la machine emploie X Window explorez les menus afin de démarrer un « terminal » (il peut-être appelé « Terminal », « Konsole », « KTerm », « Gnome terminal », « xterm », « rxvt »...), par exemple via le menu «Applications», «Accessoires» ou «Système».

Vous vous trouvez alors devant le shell, qui est un logiciel grâce auquel vous pourrez passer des commandes, afin de démarrer d'autres programmes par exemple. Plusieurs shells existent, le plus répandu est nommé bash (csh et zsh en sont d'autres).

Le shell affiche, en début de ligne, quelques caractères fournissant diverses informations et invitant à lui communiquer une commande.

C'est ce que l'on appelle son invite (synonyme : prompt).

Arrêter le système (**shutdown**) [modifier]

(Ceci ne vaut pas sous l'émulateur préconisé, toutefois c'est une nécessaire habitude à contracter)

Ne jamais éteindre brutalement la machine, il FAUT INFORMER Linux de votre souhait de l'arrêter afin qu'il prenne les dispositions nécessaires.

Pour cela invoquer, sous le compte root:

shutdown -r now

réamorce immédiatement l'ordinateur.

shutdown -h now

arrête complètement le système. Vous pouvez éteindre l'ordinateur lorsque sera affiché:

```
System halted
The system is halted
```

Variables d'environnement (echo) [modifier]

Nous traiterons d'un shell nommé bash.

Commençons par nous assurer que c'est bien le shell employé.

Saisir **echo $SHELL**.

Soumettez cette commande au système grâce à la touche « Entrée », évidemment!

Le système retourne le nom du shell courant. S'il ne s'agit pas de « /bin/bash » certains des éléments ci-après exposés ne seront pas corrects.

(Sous l'émulateur préconisé aucun nom ne sera retourné. Il fournit en fait d'« ash », assez compatible et complet pour que le plus gros de ce qui suit soit utilisable)

En réalité **echo** est la commande proprement dite, qui fait office de verbe d'action, et "$SHELL" l'argument que nous lui fournissons, qui précise sur quoi elle doit travailler.

Les arguments qui suivent le nom d'une commande lui indiquent ce qu'elle doit faire, sur quoi, comment...

La commande **echo** se contente de répéter les arguments qui lui sont fournis, de sorte que **echo toto** affiche "toto" et **echo titi tutu** affiche "titi tutu".

« $SHELL » désigne le contenu d'une variable d'environnement (c'est ce que le signe dollar placé en préfixe révèle) nommée « SHELL ». Lorsqu'il lit un caractère '$' suivi d'un groupe de caractères le shell remplace l'ensemble par le contenu de la variable d'environnement ainsi nommée.

Pour examiner le contenu d'une variable saisir **echo $*nom_de_la_variable*.**

echo $SHELL affiche le contenu d'une variable d'environnement nommée « SHELL », qui par convention recèle le nom du shell actif.

La variable **PATH** définit la liste des chemins où sont cherchés les fichiers exécutables invoqués. Pour examiner son contenu: **echo $PATH**. Le séparateur est le caractère ':'. Si le PATH contient "/usr/local/bin:/usr/bin:/bin", alors à chaque fois que le système tentera, sous votre compte, de lancer un exécutable il cherchera le fichier l'abritant tout d'abord dans le répertoire nommé "/usr/local/bin" puis, s'il ne l'y trouve pas, dans "/usr/bin", puis dans "/bin".

Si vous souhaitez ajouter au PATH un répertoire "/home/delcros/bin" qui contient vos programmes personnels, il vous faudra éditant le fichier "~/.bash_profile" de sorte qu'il contienne: "PATH=$PATH:/bin:/usr/bin:/sbin:/usr/sbin:/usr/local/bin:/home/delcros/bin". Pour que son contenu soit pris en compte après une modification vous devez vous reloguer (ou utiliser la commande « source », non exposée ici).

La variable **PS1** contient la définition du prompt: **PS1="[\u@\h \w \$] "** affichera votre nom d'utilisateur (\u), puis le caractère "@", puis le nom de la machine (\h, pour *h*ost), puis une espace, puis le répertoire de travail courant (\w, pour *w*orking directory), puis un caractère '#' si vous êtes root et '$' sinon.

Voici un autre paramétrage de prompt: PS1="[\t \d \u@\h \w \$] "

Se déplacer dans l'arborescence de répertoires (**cd**) [modifier]

L'invite est paramétrable et ressemble souvent à ceci : *[root@mistra /root]$* (*mistra* est le nom de ma machine).

Le premier **root** signifie que vous vous êtes actif sous le compte utilisateur de l'administrateur système. Vous êtes donc en pleine possession de la machine, vous pouvez faire absolument n'importe quoi, jusqu'à supprimer des fichiers nécessaires à son fonctionnement... faites donc très attention... En théorie il ne faut utiliser la machine sous ce compte qu'afin de l'administrer. Des comptes dits « d'utilisateurs » permettent de travailler en temps normal sans risquer de mettre le système en panne en cas d'action inconsidérée ou de « mauvaise manipulation ». Nous exposerons ci-après comment créer un compte utilisateur.

Dans le prompt *[root@mistra /root]$* la mention **/root** exprime que vous vous trouvez dans un répertoire nommé « /root ». Sous Unix un répertoire n'est pas essentiellement différent d'un fichier, vous lirez ou entendrez donc souvent le mot « fichier » utilisé pour désigner un objet à la nature non déterminée qui peut être un fichier ou un répertoire.

Tout fichier (ou répertoire...) se trouve à un endroit précis, appelé son chemin d'accès, constitué:

- du nom du répertoire racine qui est présent sur tous les systèmes donc commun à tous les chemins et par convention appelé « / » (« barre oblique », plus communément appelé *slash*) ou Le tout premier répertoire, , est appelé « / ». Tous les autres répertoires y prennent plus ou moins directement racine)
- suivi de la série des noms des répertoires qui y mènent, deux noms consécutifs étant séparés au moyen de la barre oblique
- suivie de son nom

Si un répertoire nommé « répertoire1 » contient un répertoire appelé « répertoire2 » recelant lui-même un fichier « monfichier » le chemin complet de ce dernier est « /répertoire1/répertoire2/monfichier ».

Actuellement, donc, vous vous trouvez connecté sous le compte de l'administrateur système et dans le répertoire appelé « /root », ou peut-être « /var/root ».

Déplaçons-nous dans la "racine" du système en saisissant *cd /*.

Séparez bien **cd** et "/" par un caractère espace car Unix exige une grande précision.

En pratique **cd** est la commande et "/" l'argument que nous lui fournissons. Les arguments suivent la commande et sont parfois précédés d'un tiret.

Vous êtes maintenant dans le répertoire racine et le prompt devient **[root@*nom_de_votre_machine* /]#**.

Tapez la commande **ls** (*l*ist) afin d'obtenir la liste de son contenu, vous devez obtenir quelque chose comme:

```
bin  boot  cdrom  dev  devfs  etc  floppy  home  initrd  lib  lost+found  mnt  opt  proc  root  sbin  sys  tmp  usr var
```

Si certains fichiers ou répertoires manquent sur votre machine ce n'est pas important.

Déplaçons-nous dans le répertoire qui contient une grande partie des programmes (souvent simplement appelés « binaires »)

nommé */usr/bin* en saisissant **cd /usr/bin**. La commande reste **cd** mais cette fois l'argument est "/usr/bin". Vous pouvez là aussi obtenir le contenu du répertoire en utilisant la commande **ls**.

Maintenant allons voir ce que contient le répertoire **/etc** (aperçu lorsque nous avons listé le contenu du répertoire racine **/**). Nous avons deux possibilités pour nous y rendre : soit nous revenons dans le répertoire racine et nous nous rendons ensuite dans le répertoire **etc**; soit nous nous rendons immédiatement dans le répertoire **/etc**.

Méthode n°1 : **cd /** (pour se rendre à la racine) puis **cd etc**

Cette méthode est fastidieuse car elle nécessite de taper deux commandes successives. Nous pouvons utiliser la deuxième méthode pour nous rendre directement dans le répertoire « /etc » en écrivant le chemin complet :

Méthode n°2 : **cd /etc**

... et nous voici directement placés dans le répertoire « /etc ». Nous avons indiqué que pour se rendre dans le répertoire « etc », il fallait d'abord se rendre dans le répertoire racine. Pour ce faire nous avons placé un « / » devant « etc ».

Lorsque l'on fournit à la commande **cd** un « ~ » au lieu d'un chemin, elle nous déplace automatiquement dans notre répertoire personnel, également appelé « répertoire utilisateur » ou « home, celui où nous pouvons stocker nos travaux. Si vous êtes l'administrateur système la commande **cd ~** vous placera dans le répertoire « /root ». Dans le cas où je suis (je suis loggé en tant qu'utilisateur « delcros ») je vais automatiquement me retrouver dans le répertoire **/home/delcros**. Les répertoires des utilisateurs non dotés de privilèges se trouvent d'ordinaire sous **/home**.

Pour revenir dans votre répertoire personnel **cd ~** est la commande orthodoxe, toutefois **cd** suffit.

À quel programme correspond une commande ? (type) [modifier]

Certaines commandes sont dites « internes » car le shell contient le programme correspondant (réagissant à la commande, par exemple « cd »).

La plupart des commandes se trouvent dans les répertoires nommés « /bin », « /usr/bin ». Les répertoires « /sbin » et « /usr/sbin » recèlent des commandes utiles à l'administrateur, et « /usr/local/bin » abrite des commandes mises au point qui ne furent pas fournies par la distribution, donc par exemple des réalisations de l'administrateur.

Lorsqu'on soumet à un shell une commande qu'il ne connaît pas il détermine s'il s'agit du nom d'un programme exécutable, et si c'est le cas le démarre.

Pour diverses raisons certaines commandes existent à la fois sous forme interne et externe. En ce cas la version interne est par défaut exécutée, sauf si l'on fournit le chemin d'accès de la version externe. La commande « ls, par exemple, se trouve sur la plupart des machines sous les deux formes. Invoquer « ls » démarre la version du shell, pour lancer la commande externe il faut par exemple invoquer « /bin/ls ».

Pour déterminer si une commande est interne ou externe fournir son nom à la commande « type ». Exemple:

```
$ type cd
cd est une primitive du shell
```

D'autres, par exemple « time », sont des mot-clés plutôt que des commandes car ne sont pas autonomes (ils doivent porter sur une commande).

Pour déterminer où se trouve, parmi les répertoires du PATH, le fichier exécutable d'un programme, par exemple d'une commande externe, employer la commande « which » (qui signifie « lequel ») suivie de son nom, par exemple « which ls ». Si aucune réponse n'apparaît le programme n'existe pas ou est une commande interne. Pour obtenir une liste complète utiliser l'option « -a »:

```
$ which -a ls
/bin/ls
```

Pour chercher un fichier indépendamment du contenu du PATH on peut employer « find » ou (s'il est installé) « locate » qui repose sur une sorte de base de données des noms de fichiers périodiquement rafraîchie (donc peut fournir un résultat obsolète).

Dans quel répertoire suis-je actuellement ? (**pwd**) [modifier]

Lorsque l'on se déplace dans les répertoires, par défaut le prompt bash n'affiche que le « nom court » de celui où l'on se trouve. Le nom court ne comprend pas le chemin complet. Or il peut arriver qu'un même nom court corresponde à plusieurs répertoires bien distincts, donc que seuls les chemins qui y mènent permettent de les distinguer. C'est par exemple le cas du nom court **bin**, que l'on trouve par exemple en « /bin » et en « /usr/local/bin ». Il existe beaucoup d'autres exemples.

Pour connaître le chemin du répertoire où l'on se trouve on utilise la commande **pwd** (*p*ath of current *w*orking *d*irectory).

Lister le contenu d'un répertoire (**ls**) [modifier]

La commande **ls** permet d'obtenir beaucoup d'informations sur les fichiers et répertoires.

Déplaçons-nous par exemple dans le répertoire « /bin »: **cd /bin**

Puis demandons liste de son contenu: **ls**

Motifs (glob) [modifier]

Nous pouvons demander à **ls** de ne lister que les fichiers dont les noms contiennent des caractères donnés. Gagnons le répertoire "/bin": **cd /bin** puis demandons par exemple la liste des seuls noms de fichiers dont les noms commencent par la lettre "g": **ls g***

L'argument "g*" que nous venons de fournir est un motif, auquel on peut penser comme à une sorte de filet dont les mailles sont définies par nous (utilisateur), en respectant un certain nombre de conventions, et qui est ensuite appliqué (*jeté*) par le shell afin de sélectionner des noms de fichiers qui y correspondent.

Les principales conventions valides dans ce contexte, donc lorsque l'on définit un motif de sélection de noms de fichiers destiné au shell, sont:

- l'étoile (ou plutôt l'astérisque) est un caractère générique sélectionnant n'importe quel ensemble de caractères
- le point d'interrogation sélectionnant n'importe quel caractère

Lorsque nous invoquons **ls g***:

1. dans un premier temps le shell applique le motif (jette le filet) et sélectionne les noms de fichiers du répertoire courant qui correspondent au motif
2. dans un deuxième temps le shell lance **ls** en lui communiquant ce qu'il a sélectionné, donc par exemple (si les seuls fichiers du répertoire courant dont le nom commence par "g" sont « grep », « gunzip », « gzexe » et « gzip »): **ls grep gunzip gzexe gzip**

Cette interprétation de motif puis sélection des noms de fichiers y correspondant est en anglais appelée "glob". Elle fonctionne pour toutes les commandes, on peut donc saisir **echo g***... et obtenir le même résultat qu'avec **ls g***.

On peut également interdire au shell d'effectuer ce « glob » en plaçant l'argument entre apostrophes. Exemple: **ls 'g*'**, qui produit ici « ls: ne peut accéder g*: Aucun fichier ou répertoire de ce type », car **ls** cherche bien entendu alors un fichier nommé "g*" (g

suivi d'un astérisque), qui n'existe pas.

echo 'g*', en revanche, fonctionne parfaitement puisque 'g*' devient alors un littéral qui sera traité par **echo** comme n'importe quelle autre chaîne... qu'il se contente donc d'afficher, indépendamment de ce que contient le répertoire courant.

Options [modifier]

Voici quelques options intéressantes de la commande **ls**:

ls -l liste les attributs de fichiers (les droits de lecture, d'écriture et d'exécution, le propriétaire, le groupe, la taille en octets, sa date de création ou de modification).

```
total 3615
-rwxr-xr-x    1 root     root         2716 Apr 23  02:09  arch
-rwxr-xr-x    1 root     root        56380 Dec 23  2008   ash
lrwxrwxrwx    1 root     root            4 May 10  20:01  awk -> gawk
-rwxr-xr-x    1 root     root        18768 Mar  8  19:17  basename
-rwxr-xr-x    1 root     root       300668 Sep  4  2008   bash
lrwxrwxrwx    1 root     root            3 May 10  19:59  bsh -> ash
-rwxr-xr-x    1 root     root        16584 Dec 16  2008   cat
-rwxr-xr-x    1 root     root        17408 Nov 26  2008   chgrp
```

(Note: la mention « total 3615 » exprime la taille cumulée des fichiers, exprimée en blocs de 512 octets.)

Ici, tous les fichiers appartiennent à l'administrateur système (root) et à son groupe (root). Nous traiterons du sens de la fin de chaque ligne, qui contient parfois une flèche visible ici sur la ligne `awk -> gawk`, dans la section consacrée aux liens ln.

-a liste tous (penser au mot "*a*ll") les fichiers du répertoire, y compris les fichiers cachés. Cette option est très utile lorsque l'on se trouve dans son répertoire personnel car il contient les fichiers de configuration de l'utilisateur dont les noms commencent généralement par un point et seule l'option **-a** permet de détecter leur existence.

D'autres options de **ls** sont utiles:

-m
: affiche en séparant par une virgule au lieu de les présenter en colonnes.

-t
: trie par date, c'est-à-dire en les classant du récent au plus ancien (penser à *t*ime).

-F
: trie par type. Ainsi un fichier suivi d'un slash (/) est un répertoire, un fichier suivi d'une étoile est un fichier exécutable et un fichier suivi d'un "@" est un lien (nous reviendrons sur les liens dans la section consacrée à **ln**).

-S
: trie par ordre de taille décroissante (penser à *s*ize).

-X
: trie par extension.

-r
: trie par ordre alphabétique inverse (penser à *r*everse). Cette option a la particularité d'inverser l'effet de toutes les autres options de tris requises. **ls -tr** affichera ainsi les fichiers par date (-t) en commençant par les plus anciens pour finir par les plus récents.

Voir le contenu d'un fichier (**less**) [modifier]

La commande **less** permet de lire le contenu d'un fichier texte. Nous avons vu que le répertoire **/root** contenait des fichiers de configuration. Ces fichiers sont simplement des fichiers textes avec un agencement et une syntaxe particulière. Regardons le contenu du fichier **.bashrc** qui permet de configurer à souhait son shell: **less .bashrc**.

Pour passer d'une page à l'autre, tapez sur la touche espace. Pour revenir sur la page précédente tapez sur « b » (*b*ack). Pour quitter tapez « q ».

Une option utile de **less** est **-N** qui numérote les lignes (la numérotation de ligne apparaît à l'écran mais le fichier n'en est pas pour autant modifié).

Si vous souhaitez connaître les autres options de **less** invoquez **less --help**.

Modifier le contenu d'un fichier (l'éditer) (**emacs**, **joe**) [modifier]

joe [modifier]

joe est simple et léger. C'est l'éditeur que j'utilise pour faire une petite modification dans un fichier de configuration.

Lançons joe: **joe**

La touche **Ctrl** (Control) permet de le commander.

Combinaisons fondamentales:

Ctrl-k h
: afficher le panneau d'aide en ligne (un nouveau **Ctrl-k h** l'effacera). Note: dans les textes d'aide de joe le caractère ^ (circonflexe) représente par convention la touche Control, donc ce que nous présentons ici par **Ctrl-k h** y est noté ^KH

Ctrl-k e
: ouvrir un fichier

Ctrl-k d
: sauvegarder

Ctrl-k x
: sauvegarder et quitter

Ctrl-c
: quitter sans sauvegarder

Attention: dans la version traduite en français de joe les questions qu'il pose sont en français... mais toute réponse qu'il doit analyser doit être fournie en anglais. Exemple: à la question « Enregistrer (o,n,^C) ? » il faut, pour accepter, répondre « y » (*y*es).

Configuration [modifier]

En tant qu'utilisateur root éditer, dans le répertoire « /etc/joe/ », les fichiers nommés « joerc » et « jmacsrc » de sorte que

- « -asis » se trouve en début de ligne (pas d'espace avant le tiret), afin de pouvoir éditer des caractères accentués
- « -backpath path » se trouve en début de ligne et soit suivi de « /tmp », la ligne devenant donc « -backpath /tmp », afin que les fichiers de sauvegarde de la précédente version d'un fichier édité se trouvent dans « /tmp » plutôt que dans le répertoire courant

Emacs [modifier]

Emacs date de la fin des années 70 et ne cesse d'évoluer depuis, ce qui fait de lui, sans aucun doute possible, l'éditeur le plus puissant.

Bien plus qu'un éditeur, il est un environnement de travail: édition, programmation, mail, news, shell ... bref on peut rester sous Emacs sans avoir besoin de quoi que ce soit d'autre.

Ses adeptes sont très nombreux, et ne leur dites pas qu'il est lourd ...vous vous tromperiez lourdement (je sais de quoi je parle ... j'ai fait l'erreur et en ce moment je suis sous Emacs... ;-))

Lançons Emacs: **emacs**

Ses combinaisons reposent sur les touches **Ctrl** (Control) et **Meta** (qui correspond souvent à **Alt**).

Ouvrons un fichier nommé « linux-test » grâce à la combinaison de touches suivante : **Ctrl-x Ctrl-f**, donc enfoncer la touche Control et la maintenir en position basse, taper « x » puis « f », et relâcher « Control ». La mention *Find File : ~/* apparaît (en bas, dernière ligne, dans le « minibuffer »). Saisir le nom du fichier et valider grâce à la touche Entrée. Son contenu apparaît et vous pouvez le modifier. Les touches **Backspace** et **Suppr** fonctionnent comme sous n'importe quel éditeur. Pour sauver tapez la combinaison de touches: **Ctrl-x Ctrl-s**.

Si vous êtes bloqué après avoir effectué de mauvaises manipulations, tapez **Ctrl-g**, si nécessaire plusieurs fois.

Si vous avez fait des erreurs, **Ctrl-x u** supprime les dernières modifications.

Si vos touches de déplacement du curseur ne fonctionnent pas: **Ctrl-p**: monter d'une ligne. **Ctrl-n**: descendre d'une ligne. **Ctrl-f**: avancer d'un caractère. **Ctrl-b**: reculer d'un caractère. **Ctrl-v**: avancer d'une page. **Alt-v**: reculer d'un écran. **Ctrl-d**: supprimer le caractère sur lequel le curseur se trouve. **Ctrl-s**: recherche incrémentale. **Meta-%**: rechercher et remplacer.

Pour quitter Emacs composer **Ctrl-x Ctrl-c**.

Il m'est impossible de décrire dans ce document les milliers de fonctions disponibles.

Exemple: si vous éditez un code source et le faites traiter (par exemple pour vérification lexicale-syntaxique, compilation... en particulier via l'utilitaire « make » ou équivalent) via Emacs, alors pour aller à la première erreur il suffit de taper C-x `. Emacs place alors votre curseur à la ligne où s'est produite l'erreur. Corrigez, puis pour aller à l'endroit correspondant à l'erreur suivante tapez encore C-x ` et ainsi de suite. Pour paramétrer d'autres combinaisons de touches:

```
(global-set-key (read-kbd-macro "C-!") 'next-error)
(global-set-key (read-kbd-macro "C-/") 'previous-error)
```

Si vous souhaitez en savoir plus, explorer ses panneaux d'aide ou des documents publiés:

- http://www.linux-france.org/article/appli/emacs
- http://www.tuteurs.ens.fr/unix/editeurs/emacs.html
- http://www.emacswiki.org/emacs-fr/NouvelUtilisateurEmacs

joe *est* Emacs [modifier]

joe peut émuler Emacs, ce qui épargne d'assimiler ses commandes propres tout en bénéficiant d'un éditeur compact et quasi partout disponible. L'invoquer pour cela avec la commande **jmacs**.

Créer un compte utilisateur [modifier]

Préambule [modifier]

Nous entrons maintenant dans une zone à risque, mieux vaut donc se connecter en tant qu'utilisateur de la machine et non pas en tant qu'administrateur système (root), car nous risquerions par une mauvaise manipulation de supprimer des fichiers fondamentaux nécessaires à son bon fonctionnement. Nous allons donc créer un compte utilisateur, lui attribuer un mot de passe et nous loguer sur ce compte.

Exécutez les commandes suivantes, une explication détaillée interviendra ensuite dans la partie consacrée à l'administration système:**adduser *le_nom_de_choix*** (votre prénom par exemple, mais sans accent et si possible long de moins de 8 caractères)

su *le_nom_de_votre_choix* (la commande **su** permet de se loguer sous un autre compte). Il suffira de saisir **exit** pour « revenir » à la session de travail root.

Invite (prompt) de root ou d'utilisateur [modifier]

L'invite proposée lorsque l'on est connecté sous compte root est souvent différente de celle d'un utilisateur sans droits étendus. Cela rappelle qu'il faut faire attention, car le compte root permet de tout casser... Traditionnellement l'invite de root est terminée par un signe dièse (#) et celle d'un utilisateur (non root) par un signe dollar ($).

Déclarer qui est propriétaire d'un fichier (**chown**) [modifier]

Préambule: cette section nécessite d'être administrateur système, il vous faut donc vous loguer sous ce compte, par exemple en utilisant la commande **su**: **[delcros@mistra ~]$ su root**

Chaque fichier appartient à un utilisateur ainsi qu'à un groupe.

Si, par exemple, nous avons effectué un **ls -l linux-test** et obtenu:
-rw-r-r-- 1 delcros users 9 Jul 19 19:03 linux-test
(Note: couleurs ajoutées pour simplifier l'exposé)
delcros est le nom du propriétaire du fichier et users est celui du groupe auquel appartient le fichier.

L'administrateur système peut décider des utilisateurs membres d'un groupe, et permettre à un utilisateur de déterminer lui même qui fera partie du groupe. Le fichier */etc/group* décrit les groupes.

Je peux décider par exemple que le fichier « linux-test » n'appartient plus à l'utilisateur "delcros" mais à l'utilisateur "thomas":

```
[root@mistra delcros]# chown thomas linux-test
```

Vérifions:

```
[root@mistra delcros]# ls -l linux-test
-rwxrw-r-- 1 thomas users 9 Jul 19 19 : 03 linux-test
```

Pour changer à la fois le propriétaire afin de le donner à *olivier* et le groupe (ici: *testeurs*): **chown olivier.testeurs linux-test**

Pour ne changer que le groupe: **chown .testeurs linux-test**

Copier un fichier ou un répertoire (**cp**) [modifier]

La syntaxe est: **cp [option] nom-fichier(s)-origine nom-fichier-ou-répertoire-destination**

Par exemple pour faire une copie d'un fichier du répertoire où l'on se trouve nommé « linux-test » en un fichier « linux-test2 »:

```
cp linux-test linux-test2'''
```

ATTENTION: si vous effectuez une copie d'un fichier sur un autre qui existe déjà, le contenu de ce dernier sera remplacé par celui de l'autre!

Si vous souhaitez copier le fichier *linux-test* dans un répertoire (par exemple « /tmp »):

```
cp linux-test /tmp'''.
```

Pour lui donner un autre nom:

```
cp linux-test /tmp/linux-test2'''
```

Options [modifier]

-i

avertit de l'existence d'un fichier du même nom et demande s'il peut ou non remplacer son contenu. Recopions à nouveau le fichier « linux-test » sur « linux-test2 », cette fois avec l'option **-i**: **cp -i linux-test linux-test2**. Le message *cp: overwrite `linux-test2'?* (ou, en version française, *cp: écraser `linux-test2'?*) apparaît. Répondre par "y" (yes) ou "n". En version française "o" (oui) fonctionne.

-b

comme l'option -i s'assure que la copie n'écrase pas un fichier existant. Le fichier écrasé est toutefois alors sauvegardé, renommé avec un suffixe tilde (~).

-v

affiche les noms des fichiers copiés (pensez à *v*erbeux). Utile si vous souhaitez vérifier le bon déroulement.

-p

lors de la copie les attributs des fichiers seront préservés (propriétaire, groupe, date de création... lire chown).

-r [modifier]

L'option **-r** copie de manière *r*écursive, donc l'ensemble d'un répertoire et de ses sous-répertoires.

Exemple: je possède dans mon répertoire « /home/delcros/personnel » un répertoire intitulé « mygale » qui contient 3 sous répertoires (« echecs », « linux », « xcaissa »):

```
/home/delcros/personnel/
/home/delcros/personnel/mygale/
/home/delcros/personnel/mygale/echecs/
/home/delcros/personnel/mygale/linux/
/home/delcros/personnel/mygale/xcaissa/
```

Je copie le répertoire « mygale » ainsi que ses sous-répertoires dans mon répertoire « /home/delcros/ » en commençant par me placer dans le répertoire recelant l'ensemble: **cd /home/delcros/personnel/** puis en le recopiant: **cp -r mygale /home/delcros**.

En associant l'option **-v** et **-r**: **cp -rv mygale /home/delcros**:

```
mygale -> /home/delcros/mygale
mygale/index.html -> /home/delcros/mygale/index.html
mygale/logo.gif -> /home/delcros/mygale/logo.gif
mygale/linux -> /home/delcros/mygale/linux
mygale/linux/linux.html -> /home/delcros/mygale/linux/linux.html
... ce n'est qu'une partie du résultat ...
```

J'ai par exemple déjà copié dans le répertoire d'un utilisateur (pour qu'il en dispose) de la documentation qui se trouvait dans un répertoire « doc » dont le propriétaire était l'administrateur système. Pour cela j'ai copié tout le répertoire et ses sous-répertoires dans le répertoire de l'utilisateur grâce à la commande **cp** et à son option **-r**, puis j'ai déclaré l'identité du nouveau propriétaire de la copie et de ses sous répertoires: **[root@mistra delcros]# chown -R delcros.users doc**. L'option **-R** (*r*ecursive) de **chown** le contraint à propager son action dans les sous-répertoires, établissant de cette seule commande le propriétaire de plusieurs répertoires et de leurs fichiers.

Déclarer les droits d'accès (permissions) accordés (**chmod**) [modifier]

Linux permet de spécifier les modes d'accès (à un fichier ou répertoire) autorisés, que l'on appelle droits d'accès ou « permissions ».

Pour voir ces droits, il suffit d'utiliser la commande **ls -l nom_du_fichier**, qui produira par exemple:

```
[delcros@mistra ~]$ ls -l bidule
-rw-r--r-- 1 adelcros users 3619 jun  6 21:49 bidule
```

Il y a là de nombreuses informations, certaines ne seront décrites que plus tard.

Le nom du fichier, « bidule », apparaît à la fin de la ligne.

Le premier « adelcros » signifie que le fichier appartient à l'utilisateur « adelcros ». La mention « users » signifie qu'il appartient au groupe « users ».

La partie qui contient *-rw-r--r--* révèle la nature du fichier ainsi que les permissions (droits d'accès).

Le premier de ces 10 caractères correspond à la nature du fichier. Un tiret signifie qu'il est tout ce qu'il y a de plus classique. Si, à la place du premier tiret, on observait un *d* cela signifierait qu'en réalité le fichier est un répertoire (*d*irectory). Si on observait un *l*, alors le fichier serait un lien.

Décomposons les 9 derniers caractères en 3 segments: **rw-** , **r--** et **r--**

1. Le premier révèle les droits accordés au propriétaire du fichier
2. Le deuxième révèle les droits accordés aux comptes qui ne sont pas celui du propriétaire et faisant partie du groupe auquel appartient le fichier
3. Le troisième révèle les droits accordés aux autres comptes

Dans chaque segment le premier caractère correspond au droit de lecture (noté *r*, penser à *r*ead), le deuxième correspond au droit d'écriture (*w*rite) et le troisième correspond au droit d'exécution (*x*, execution). Si à la place d'un des ces caractères nous ne voyons qu'un tiret -, c'est que le droit correspondant n'est pas accordé.

« -rw-r--r-- », par exemple, exprime que tous les utilisateurs ont le droit de lire le fichier et que seul son propriétaire a le droit de le modifier. Personne ne peut lancer l'exécution de ce fichier puisque *x* n'apparaît pas.

Lorsque l'on traite de permissions la lettre *u* (*u*ser) désigne par convention le propriétaire du fichier, la lettre *g* le *g*roupe d'utilisateurs qui possède le fichier et la lettre *o* (*o*thers) les autres utilisateurs. La lettre *a* (*a*ll) fait référence à tous les utilisateurs. Cette notation est la plus facile à employer avec la commande de modification des permissions **chmod**.

Seul le propriétaire du fichier (ainsi que "root", qui dispose des pleins pouvoirs) a le pouvoir de modifier ces permissions.

Établissons par exemple que n'importe qui pourra lire et modifier un fichier nommé « linux-test »:

```
[delcros@mistra delcros]$ chmod a+wr linux-test
```

- **chmod** est la commande *ch*angeant les « *mod*es d'accès »
- *a* indique que tous (*a*ll) les segments seront touchés par la modification des permissions
- + signifie que l'on accorde une ou plusieurs permission(s) supplémentaire(s), qui s'ajoute à celles qui sont déjà établies. Pour en retrancher il suffit d'utiliser en lieu et place le signe -. Pour établir les permissions indépendamment de ce qu'elles sont déjà il suffit d'employer le signe =.
- *w* signifie qu'il s'agit de la permission d'écriture (*w*rite)
- *r* signifie qu'il s'agit également de la permission de lecture (*r*ead)

Pour vérifier que tout a bien fonctionné, faites un **ls -l linux-test** qui montrera une première colonne contenant *-rw-rw-rw-*

Pour supprimer les droits d'écriture et de lecture accordés aux membres du groupe propriétaire du fichier ainsi qu'aux autres utilisateurs: **chmod go-wr linux-test**

- dans *go* le *g* signifie que la commande affectera les permissions accordées au *g*roupe propriétaire, et le *o* qu'elle affectera également celles que l'on accorde aux « autres utilisateurs » (ceux qui ne sont ni le propriétaire, ni membres du groupe auquel le fichier appartient, *o*ther users)
- *wr* signifie que la modification portera sur les droits d'écriture ou de lecture. On aurait pu aussi écrire la commande en mettant **rw** car l'ordre n'a pas d'importance.

Si je souhaite que le propriétaire du fichier ne puisse pas exécuter un fichier:

```
[delcros@mistra delcros]$ chmod u-x linux-test
```

Pour définir d'un seul mouvement toutes les permissions d'un fichier, on peut utiliser la syntaxe suivante (nous voulons que « linux-test » soit en lecture, en écriture et en exécution pour le propriétaire, que le groupe n'ait le droit que de le lire et exécuter, et que les autres utilisateurs ne disposent d'aucun droit):

```
[delcros@mistra delcros]$ chmod u=rwx,g=rx,o= linux-test
```

Pour ôter le droit d'y écrire à tous sauf au propriétaire:

```
chmod g-w,o-w linux-test
```

Lorsqu'un programme (quel qu'il soit, y compris le shell) tente d'accéder à un fichier, le noyau Unix sait pour quel utilisateur le programme fonctionne. Si le fichier appartient à l'utilisateur les droits accordés au propriétaire s'appliquent. Sinon, si le fichier appartient au groupe auquel l'utilisateur appartient, alors les droits accordés au groupe auquel le fichier appartient s'appliquent. Sinon les droits accordés aux autres utilisateurs seront respectés. Exemple:

```
adelcros@mira:~$ ls -l secret.txt
----r--r-- 1 adelcros users 362 sep 17 22:55 secret.txt
adelcros@mira:~$ less secret.txt
less: secret.txt: Permission non accordée
```

Il n'est pas stupide d'interdire ainsi l'accès alors que l'utilisateur peut modifier les permissions du fichier (exemple: **chmod u+r important.txt**) donc s'en accorder le droit, car cela constitue une sorte de garde-fou en cas d'erreur: un logiciel tentant d'accéder au contenu de ce fichier se le verra refuser et devra, pour obtenir satisfaction, modifier ses permissions, ce qui ne sera pas fait par hasard.

Autre exemple:

```
adelcros@mira:~$ ls -l important.txt
----r--r-- 1 adelcros users 362 sep 17 22:55 important.txt
adelcros@mira:~$ rm important.txt
/bin/rm: détruire un fichier protégé en écriture fichier régulier `important.txt'?
```

Expression numérique des permissions (système octal) [modifier]

Il n'est pas nécessaire d'assimiler cette section, elle n'est destinée qu'à éclairer des choses étranges parfois rencontrées, par exemple dans des documentations.

Une autre façon de déclarer (définir) les permissions existe. Elle repose sur une notation numérique remplaçant, d'une part par un 0 (zéro) chaque droit non accordé (ceux que **ls** révèle au moyen d'un tiret) et, d'autre part, les autres (accordés!) par un 1 (chiffre *un*).

Pour chaque fichier les trois segments de permission (le premier décrit les droits du propriétaire, le deuxième ceux des membres du groupe auquel appartient le fichier, le troisième ceux de tous les autres comptes) rassemblent chacun 3 droits (r, w, x), il y a donc 2^3 donc 8 combinaisons possibles. C'est pourquoi un système numérique de notation des permissions parfois adopté est dit octal.

```
  Binaire -- Logique ----- Décimal
000 -------- (---) ------- 0
001 -------- (--x) ------- 1
010 -------- (-w-) ------- 2
011 -------- (-wx) ------- 3
100 -------- (r--) ------- 4
101 -------- (r-x) ------- 5
110 -------- (rw-) ------- 6
111 -------- (rwx) ------- 7
```

'*chmod* digère cette notation. C'est pourquoi **chmod u=rwx,g=rw,o=r *nom_de_fichier*** équivaut à **chmod 764 *nom_de_fichier*** car *rwx* (droits du propriétaire) correspond à 7, rw (droits des membres du groupe auquel le fichier appartient) correspond à 6, et r (droits des autres comptes) correspond à 4. Les chiffres doivent être exprimés dans l'ordre, le premier pour le propriétaire, le deuxième pour le groupe, le troisième pour les autres.

Supprimer un fichier (*rm*) [modifier]

Effectuons à nouveau une copie du fichier « linux-test » (tapez au préalable **cd** pour vous retrouver dans votre répertoire personnel): **cp linux-test linux-test2**

Pour supprimer le fichier "linux-test2": **rm linux-test2**

Options [modifier]

-i
comme pour **cp**, l'option **-i** permet à **rm** de demander à l'utilisateur s'il souhaite vraiment supprimer le ou les fichiers en question.

-r
supprime un répertoire et ses sous répertoires (attention, c'est TRÈS dangereux)

-f
supprime les fichiers dans lesquels il est interdit d'écrire ainsi que les répertoires, sans demande une confirmation (à utiliser avec précaution...)

Créer un répertoire (**mkdir**) [modifier]

Pour créer un répertoire, il suffit de taper la commande suivante (ici je crée dans « /home/delcros » un répertoire nommé "personnel"): **mkdir personnel**

Une option de **mkdir** est souvent utile: **mkdir -p** permet de créer une suite de répertoire.

Supposons que je veuille créer dans mon répertoire **/home/delcros** la suite de répertoires suivante : **doc/mygale/mail**. Je peux faire soit: **mkdir doc** , **cd doc** , **mkdir mygale** , **cd mygale** , **mkdir mail**

Ou bien utiliser l'option **-p** qui me permet de créer la suite de répertoires "parents" le plus simplement du monde: **mkdir -p doc/mygale/mail**

Déplacer ou renommer un fichier (**mv**) [modifier]

Pour comprendre la commande **mv** (penser à *move*), voyons une suite de commandes qui effectuent des opérations différentes:
mv linux-test perso renomme le fichier "linux-test" en "perso"

mv perso perso écrase le fichier existant avec le fichier copié (dit « source »).

mv personnel mon-répertoire: renomme le répertoire « personnel » en « mon-répertoire ».

mv perso /home/delcros/mon-répertoire: déplace le fichier « perso » dans le répertoire **/home/delcros/mon-répertoire**

Options [modifier]

-b
effectue une sauvegarde (penser à *b*ackup) des fichiers avant de les déplacer: **mv -b mon-répertoire/perso /mon-répertoire/linux-test** renomme le fichier « perso » en « linux-test », cependant vous trouverez dans le répertoire une sauvegarde de perso nommée « perso~ ».

-i
demande pour chaque fichier et chaque répertoire s'il peut ou non déplacer fichiers et répertoires (penser à *i*nteractif)

-u
demande à mv de ne pas supprimer le fichier si sa date de modification est la même ou est plus récente que son remplaçant (penser à *u*pdate).

Chercher un fichier ou répertoire (**find**, **locate**, **which**) [modifier]

find [modifier]

Exemple simple: comment trouver un fichier portant un nom donné ? **find /home -name linux-test2**
Analyse:

"/home"
indique que nous voulons chercher à partir du répertoire « /home »

"-name"
indique que nous voulons trouver un fichier portant un nom donné.

Pour chercher tous les fichiers commençant par "linux-tes" et définir à partir de quel répertoire on souhaite effectuer la recherche on utilise cette syntaxe: **find /home/delcros -name 'linux-tes*'**

Le nombre d'options de **find** est impressionnant. En voici quelques unes:

-type [modifier]

Indique le type de fichier que l'on recherche. Si vous cherchez seulement un répertoire vous pourrez utilisez cette option ainsi: **find /usr -type d ...**. Ici, on demande à **find** de trouver des répertoires, et non des fichiers, grâce à l'option **-type d** (penser à *d*irectory, qui signifie *répertoire*)

-exec [modifier]

Exécute une commande sur les fichiers trouvés.

La différence entre **-exec** et **-ok** est que la deuxième vous demandera pour chaque fichier trouvé si vous souhaitez réellement réaliser l'opération: **find -name 'linux-tes*' -print -ok rm {} \;** `./linux-test`
`rm ... ./linux-test ? y`

Avec l'option **-exec** la paire d'accolades indique l'endroit où se trouvera, lors de l'invocation de la commande, le nom du fichier trouvé. L'anti-slash lié au point virgule forme une séquence d'échappemment.

locate [modifier]

La commande **locate** a la même mission que **find** mais explore une base de données qui contient la liste des fichiers et répertoires. Cette base de données est en général automatiquement mise à jour quotidiennement par le système.

La syntaxe est: **locate nom_du_fichier**.

Bien que la commande **locate** soit très intéressante, elle ne possède pas la puissance des options de **find**. De plus, si vous créez des fichiers pendant la journée elle ne trouvera pas avant mise à jour de sa base de données (le lendemain).

Trouver un texte (**grep**) [modifier]

La commande **grep** est un pivot des commandes UNIX. Elle cherche une expression regulière dans un ou plusieurs fichiers.
Exemple: **grep fouille linux-commande.html** cherche les lignes contenant le mot "fouille" dans le fichier nommé « linux-commande.html ».

L'option **-n** produira le numéro de chaque ligne contenant un résultat. Une autre option très utile est **-l** qui permet de n'afficher que les noms des fichiers contenant ce que l'on cherche.

Compactage et décompactage (**compress**, **gzip**, **bzip2**, **7zip**) [modifier]

compress [modifier]

Extension d'un nom d'un fichier compacté: **.Z**

Il est obsolète, ne pas compacter grâce à lui.

Si vous rencontrez un fichier d'extension **.Z** vous pouvez le décompacter grâce à gzip: **gzip -d *nom_du_fichier*.Z**

gzip [modifier]

Extension d'un nom d'un fichier compacté: **.gz**.

Efficace et très répandu.

Pour compacter un fichier: **gzip nom_du_fichier** , cela créera un fichier **nom_du_fichier.gz** et détruira l'original.
Pour décompacter: **gzip -d *nom_du_fichier*.gz** , qui recréera l'original et supprimera la version compactée.

bzip2 [modifier]

Extension d'un nom d'un fichier compacté: **.bz2**.

Sa syntaxe et son comportement correspondent à ceux de gzip.

Pour compacter un fichier: **bzip2 nom_du_fichier**
Pour décompacter: **bzip2 -d *nom_du_fichier*.bz2**

7zip [modifier]

Archivage de données (**tar**) [modifier]

La commande tar gère des archives, contenant chacune au moins un répertoire ou fichier.

Vous aurez souvent besoin de **tar xzf *nom_du_fichier*.tar.gz** , qui décompacte une archive au format **.tar.gz** ou **.tgz**.

L'extension **.tar.gz** indique que le fichier est une archive tar et qu'il est compacté (**.gz** révèle qu'il l'est avec gzip).

Les arguments employés dans la commande précédente (« xzf ») peuvent être ainsi compris:

x
: (**x**traction) déclenche l'extraction de certains fichiers d'une archive (lorsque l'on ne spécifie pas les noms des fichiers que l'on souhaite extraire de l'archive, **tar** les extrait tous)

z
: décompacte l'archive grâce à gzip

f
: traite un **f**ichier-archive dont le nom suit (ici: « *nom_du_fichier*.tar.gz »)

Pour connaître la liste des fichiers contenus dans une archive compactée on emploiera **tar tvzf *nom_du_fichier*.tar.gz**

t
: dresse liste du contenu de l'archive

v
: est le mode "*v*erbeux", qui montre tout

Si je me trouve dans le répertoire « « /home/delcros/ » » la commande suivante créera une archive du répertoire « /home/delcros/personnel »: **tar cvjf personnel.bz2 personnel**

c
: tar va *c*réer une archive

j
: tar compacte l'archive grâce à bzip2

Connaître l'espace disque utilisé (**df** et **du**) [modifier]

La commande **df** permet de connaître l'emplacement de montage des systèmes de fichiers accessibles sur votre système et leurs capacités restantes.

La commande **du** permet de connaître l'encombrement sur disque du répertoire spécifié et ses sous répertoires.

Options [modifier]

-c
: total *c*umulé

-h
: adoption d'unités facilitant la lecture du résultat par un *h*umain

Contrôler les ressources utilisées par les processus (**top**, **ps**, **kill**) [modifier]

top [modifier]

La commande **top** affiche en continu des informations décrivant l'activité du système. Elle permet surtout de suivre les ressources que les processus utilisent (quantité de mémoire, pourcentage de CPU...).

Vous pourrez utiliser l'option **-d** pour spécifier des délais de rafraîchissement (en secondes).

Sous **top** il est possible d'expédier de manière interactive un signal à un processus, par exemple afin de le stopper, en tapant **k**.

top demande ensuite quel signal il doit envoyer et 15 (SIGTERM) est le signal par défaut qui met fin à un processus, 9 (SIGKILL) est beaucoup plus brutal et à éviter.

Pour quitter **top**, appuyer simplement sur la touche "q".

Une famille d'utilitaires de type *top* existe.

ps [modifier]

La commande **ps** dresse liste des processus actifs: **ps**

```
PID   TTY STAT  TIME COMMAND
341   p1    S     0:00    bash
344   p2    S     0:00    bash
1039  p3    S     0:00    bash
1219  p3    R     0:00    ps
```

Le PID est l'identificateur d'un processus, c'est un nombre. Chaque processus est identifié dans le système par un nombre unique. Le "TTY" indique à quel port de terminal est associé le processus. "STAT" indique l'état (*stat*us) dans lequel se trouve le processus.

Dans l'exemple, trois processus sont endormis (S comme "sleep"), et un autre en cours d'exécution (R comme "run"). Ce dernier n'est autre que la commande **ps** que nous venons de lancer, qui se « voit elle-même ». Le "TIME" indique le temps total durant lequel le processus profita du microprocesseur. "COMMAND" précise, comme son nom l'indique, la commande qui créa le processus.

Une simple commande **ps** n'indique pas tous les processus du système. Le simple fait de lancer **ps** nous a juste indiqué les processus associés à un terminal qui dépendent de l'utilisateur courant. En fait, il est tout a fait probable que d'autres processus non liés à un terminal soient actifs: **ps -ax** les révélera.

Si en plus vous voulez connaître les utilisateurs associés à chaque processus, il vous suffit d'utiliser l'option **aux**. Vous verrez alors plusieurs colonnes s'ajouter dont "USER" qui indique à quel utilisateur appartient le processus. "%CPU" indique en pourcentage les ressources du microprocesseur sur le moment utilisées par le processus. "%MEM" montre en pourcentage la proportion de mémoire vive dans la mémoire utilisée par le processus, elle-même fournie dans la colonne "RSS". "START" indique l'heure à laquelle le processus a été lancé.

pstree [modifier]

Cette commande permet d'afficher les processus sous forme d'arborescence donc de voir leurs inter-dépendances: **pstree**

```
init-+-crond
|-emacs---emacsserver
|-gpm
|-inetd
|-kerneld
|-kflushd
|-klodg
|-kswapd
|-loadmeter
|-lpd
|-6*[mingetty]
|-named
|-netscape---netscape
|-4*[nfsiod]
|-nxterm---slrn-gor---slrn
|-portmap
|-pppd |-rc.news---innwatch---sleep
|-rpc.mountd
|-rpc.nfsd
|-rpc.yppasswdd
|-sendmail
|-syslogd
|-update
|-xconsole
|-xdm-+-X
| `-xdm---Xsession---fvwm---FvwmPager
|-xterm---bash---su---bash---tail
|-2*[xterm---bash]
|-xterm---bash---pstree
\ -ypserv
```

On voit par exemple ici que j'utilise « Fwmpager » qui a lui-même été lancé par « fwm ».

kill [modifier]

La commande "**kill**" permet d'expédier un signal à un processus.

Sa syntaxe est la suivante: **kill [options] PID**

Elle est souvent utilisée afin d'interrompre un programme sans IHM ou récalcitrant. Si j'ai lancé un programme donné, par exemple *xeyes*, un processus sera actif. Si je le lance deux fois, deux processus seront actifs, et ainsi de suite.... Pour demander à un processus de cesser de fonctionner (on parle de le «tuer»), je peux d'abord invoquer **ps auxw** pour connaître son PID, si par exemple le PID est 592, le tuer en invoquant **kill 592**

```
[delcros@mistra ~]# ps auxw|grep xeyes
    592  0.6  0.0   5884  1792 pts/2    S    10:53   0:00 xeyes
    596  0.0  0.0   3940   780 pts/2    S+   10:53   0:00 grep xeyes
```

La première colonne présente le PID et la dernière le nom du programme invoqué. La seconde ligne, ici, correspond au *grep* invoqué, qui se «voit ainsi lui-même».

Seul l'utilisateur auquel le processus appartient (ou root) peut le tuer.

La commande **killall** tue un processus mais au lieu de fournir son PID vous indiquerez son nom (la commande qui le lança). Si plusieurs processus y correspondent vous tentez ainsi de tous les tuer, mieux vaut utiliser l'option "-i" (pour interactif) qui demande confirmation avant de tenter d'arrêter un processus.

Un processus peut se trouver dans un état ne permettant pas de lui demander de s'interrompre. Tenter en ce cas de lui expédier le signal HUP puis INT en invoquant par exemple "kill -HUP *PID*". S'il fait la sourde oreille demander au noyau de le tuer sans condition en lui expédiant KILL («KILL» est ici le libellé d'un signal). Attention: le signal KILL ne laisse pas le processus ainsi condamné prendre des dispositions parfois souhaitables ou nécessaires, par exemple fermer un fichier, donc peut endommager ou perdre des données voire causer des dysfonctionnements.

Redirection (des entrées/sorties d'exécutables) [modifier]

Lorsque vous saisissez des données à mesure reçues par un processus vous fournissez des éléments à traiter, appelés entrées, grâce au clavier. L'activité des processus produit des sorties (résultats), le plus souvent par défaut affichés.

L'entrée de données peut se faire par un canal autre que le clavier, en indiquant par exemple un fichier qui contient les données à traiter. Le canal de sortie peut lui aussi être dirigé ailleurs que vers l'écran, par exemple vers un fichier dont on pourra ensuite continuer a traiter le contenu.

On peut également recevoir ou expédier des données à chacun des périphériques connectés, auquel correspond un « fichier spécial » dans le répertoire « /dev ».

Le shell offre un moyen de gérer tout cela, en interconnectant à loisir ces canaux, grâce aux redirections.

Comment faire, par exemple, pour que la commande **less**, qui affiche le contenu d'un fichier, le place plutôt dans un fichier? C'est

le signe **>** qui va nous permettre cela.

Dans un premier temps, je veux voir le contenu du fichier nommé « test »: **less test**. La commande **less** affiche son résultat vers la sortie standard qui est le terminal.

Dans un deuxième temps, je veux placer le fichier « test » dans un fichier « test2 »: **less test > test2**. Ce n'est qu'une copie du fichier "test" en "test2". La commande **cp** permet aussi de le faire.

Plusieurs types de redirections existent:

> nom_de_fichier
: redirige vers le fichier nommé ce que produit le programme. Exemple: **ls > ls.txt**

>> nom_de_fichier
: permet comme le signe ">" de rediriger le canal de sortie standard vers le fichier nommé. Si ce dernier existe déjà, ce qui provient du canal de sortie sera ajouté après ce qui y existe déjà alors qu'avec un simple ">" le contenu existant serait remplacé (écrasé)

< nom_de_fichier
: redirige le canal d'entrée standard, qui devient le contenu du fichier nommé. Exemple: **less < ls.txt** . Cela revient apparemment à ce que montre **less ls.txt** mais est subtilement différent car interdit à **less** d'accéder directement au fichier en demandant au shell d'en copier le contenu sur le canal d'entrée standard de **less**, qui ne pourra savoir d'où il vient

&> nom_de_fichier
: redirige sortie standard et erreur standard vers un seul et même fichier

Ce sont les formes les plus fréquemment employées.

Toutes ces formes sont des versions simplifiées. Pour maîtriser tout cela il faut savoir que l'entrée standard dans un système Unix est le clavier, désigné par le descripteur de fichier numéroté "0". Par défaut le terminal est la sortie standard, ce descripteur de fichier est désigné par le chiffre "1". Il existe un troisième descripteur de fichier classique qui est le canal de sortie des erreurs produites par l'exécution d'une commande, désigné par le chiffre "2". Dans les exemples suivant la lettre *n* remplace l'un de ces chiffres:

n> nom_de_fichier
: redirige ce qui sort via le descripteur *n* (vers le fichier nommé). Par exemple, si vous souhaitez obtenir les erreurs standards (donc le canal 2) dans un fichier nommé *monprogramme.err*: ***nom_de_commande* 2> monprogramme.err**

n< nom_de_fichier
: déclare le fichier nommé comme correspondant au descripteur

>&n
: duplique la sortie standard vers le fichier correspondant au descripteur *n*

<&n
: duplique l'entrée standard depuis le fichier correspondant au descripteur *n*

À première vue, on se demande bien à quoi servent certaines des redirections... On le découvre en employant le shell.

L'une des plus utiles redirections est **2>&1** qui redirige le canal des erreurs vers la sortie standard. Elle est très appréciée des utilisateurs lorsqu'ils souhaitent conserver la liste des messages d'erreur produits par un programme: ***nom_du_programme* 2>&1 erreurs*nom_du_programme*.tmp**

Les canaux sont autant de tuyaux véhiculant les données, que l'on peut assembler librement grâce aux redirections fournies par le shell. Cela mène à la plomberie...

Interconnexion de commandes: les pipes [modifier]

Le mot "pipe" désigne la barre verticale ("|") obtenue avec la combinaison de touches "AltGr + 6" sur les clavier français classiques, ou "Altgr + 1" sur les claviers franco-belges.

Sur la ligne de commande il permet de créer des « tubes ».

Un tube passe le résultat d'une commande à une autre, sans passer par une redirection qui impliquerait un fichier temporaire.

Un exemple permettra de comprendre tout cela.

Je veux savoir quels sont les processus "bash" qui fonctionnent sur le système, mais je veux que la commande **ps aux** ne me fournisse que les lignes qui contiennent le mot "bash" pour m'épargner de lire tout ce qu'elle produit.

Avec ce que nous connaissons déjà il nous faudrait une première commande visant à obtenir la liste des processus "bash" actifs et à la placer dans un fichier temporaire: **ps aux > temporaire.tmp** . Puis nous pourrions mobiliser grep: **grep bash temporaire.tmp** afin d'obtenir le résultat souhaité. Il ne resterait plus qu'à nettoyer: **rm temporaire.tmp**.

Employer un tube réalise l'équivalent en sollicitant moins le système, qui n'aura rien à stocker temporairement: **ps aux | grep bash**

On peut dire que l'on a lancé un traitement qui connecte les deux commandes (**ps** et **grep**) entre elles, de sorte que ce que la première citée produit soit immédiatement communiqué à l'autre. Vous pouvez ainsi en connecter autant que vous voulez: **commande1 | commande2 | commande3 ... | commandeN**

Si on prend conscience de l'utilité des tubes, on les utilise et on finit par ne plus vouloir s'en passer.

Lien (**ln**) [modifier]

Un lien est un type spécial de fichier qui permet à plusieurs noms de fichiers de faire référence au même contenu sur le disque. Ce sont des sortes de « synonymes », d'aliases.

On en distingue deux sortes: « dur » et « symbolique ».

Un lien *dur* associe deux ou plusieurs fichiers à un même espace sur le disque, en préservant toutefois leurs indépendances lors de modifications de leurs contenus.

Exemple: créons un fichier (dont le contenu importe peu) nommé *linux-test* grâce à **echo toto > linux-test**

Puis créons un lien dur vers ce fichier: **ln linux-test linux-test-lien-dur**

Le fichier « linux-test-lien-dur » est créé dans le répertoire courant. Si vous faites un **ls -l** vous constaterez que « linux-test » et « linux-test-lien-dur » ont la même taille. Ils semblent indépendants mais en réalité un seul contenu (commun aux deux) existe pour le moment. Si le fichier « linux-test-lien-dur » est modifié, nous aurons automatiquement une modification du fichier « linux-test » (et vice-versa), car elle s'effectuera physiquement sur le disque dur à l'endroit stockant le contenu des deux fichiers.

Si nous créons un lien *symbolique*: **ln -s linux-test linux-test-lien-symb**

ls -l montrera le nom du fichier « linux-test-lien-symb » suivi d'une flèche révélant qu'il pointe sur « linux-test ». Toute référence au fichier « linux-test-lien-symb » impliquera en réalité le fichier « linux-test ».

En résumé un lien *symbolique* est un fichier de petite taille qui a une existence propre et contient le nom du fichier auquel il correspond.

Dans le cas d'un lien *dur*, la suppression de l'un des deux fichiers n'affectera pas l'autre. Dans le cas d'un lien symbolique, la

suppression du fichier originel (la cible du lien) rendra le lien symbolique inutilisable car il ne correspondra plus à un fichier valide et sera donc dit "cassé" (pendouillant).

Les liens sont utiles si vous souhaitez qu'un fichier apparaisse dans plusieurs répertoires, ou sous un nom différent. Si le fichier est encombrant une copie par **cp** entraînerait un gâchis alors qu'un lien permettra de réduire l'utilisation d'espace disque. Mieux: n'établir que des liens (au lieu de copier) garantit que toute modification concernera toutes les « copies » dispersées.

Syntaxe: **ln nom-fichier-source nom-fichier-lien**
Arguments:

-s
: crée un lien symbolique. Dans la plupart des cas c'est ce qui est adéquat, le lien dur est en fait rarement employé.

-b
: réalise une sauvegarde d'un fichier existant et dont nous aurions utilisé le nom, avant de l'écraser.

-i
: demande à l'utilisateur, si le lien existe, s'il souhaite l'écraser.

Alias (**alias**) [modifier]

Lorsque l'on utilise les mêmes commandes avec de nombreuses options fixes il est utile de remplacer le tout par un ordre très court déclaré comme une sorte de synonyme. Il est appelé « alias ».

Les alias sont déclarés dans le fichier « ~/.bashrc_aliases » (lui-même sourcé par « ~/.bashrc »). Voici un exemple classique de déclaration d'alias: **alias ll="ls -NlF --color=auto"**, qui crée une commande **ll** appelant **ls** avec des options améliorant la présentation.

La variable "$*" désigne tous les paramètres communiqués lors de l'appel de l'alias:

```
$ alias perroquet="echo $*"
$ perroquet ceci est un test
ceci est un test
$ perroquet autre bidule
autre bidule
```

Plutôt que de saisir souvent **tar -tvzf *nom_de_fichier*** ou pourra ainsi préférer créer un alias: **alias montar="tar -tvzf $*"** , puis employer **montar *nom_de_fichier***

La commande **unalias** supprime de façon définitive un alias. Son option **-a** (pour **all**) commande suppression de tous les alias.

Certains alias sont déclarés dans des fichiers excutés par bash lors de son démarrage, par exemple « ~/.bashrc ». En ce cas il faut commenter la ligne définissant l'alias dans le dit fichier pour le supprimer définitivement, sinon il ne le sera que durant la session en cours.

On peut automatiser de façon plus ambitieuse.

Pour aller plus loin [modifier]

La commande **man** est utile. Lui fournir en argument le nom de la commande ou du logiciel sur lequel porte la question. Exemple: **man cp**. Pour quitter le **man** appuyer sur la touche "q".

- Arborescence Linux
- Introduction à la programmation en bash
- En cas de blocage, problème...

Catégories : bash | pour débutant | premier auteur:Armand Delcros

GNU FDL FREE DOC LICENSE
Dernière modification de cette page le 19 décembre 2013 à 12:16.
Sous licence GNU Free Documentation License 1.3 Politique de confidentialité À propos de Linux France Avertissements
Powered By MediaWiki